LAB 4: GAUSSIAN PROCESSES

Namita Sharma

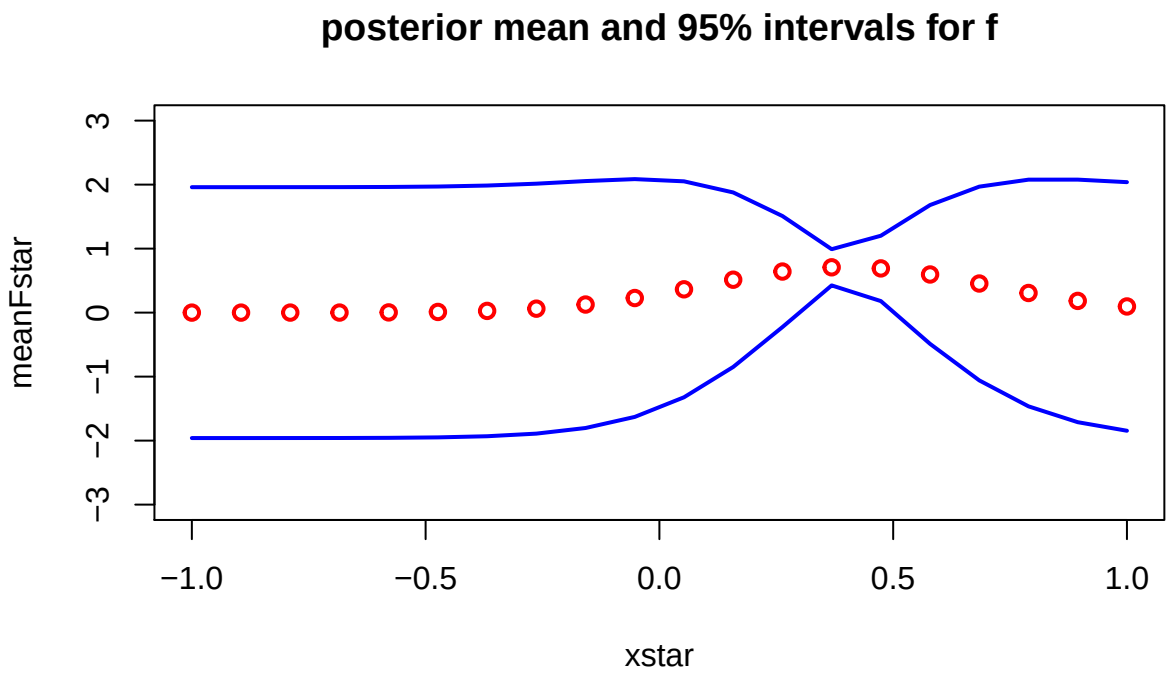
10/16/2020

1. Implementing GP Regression

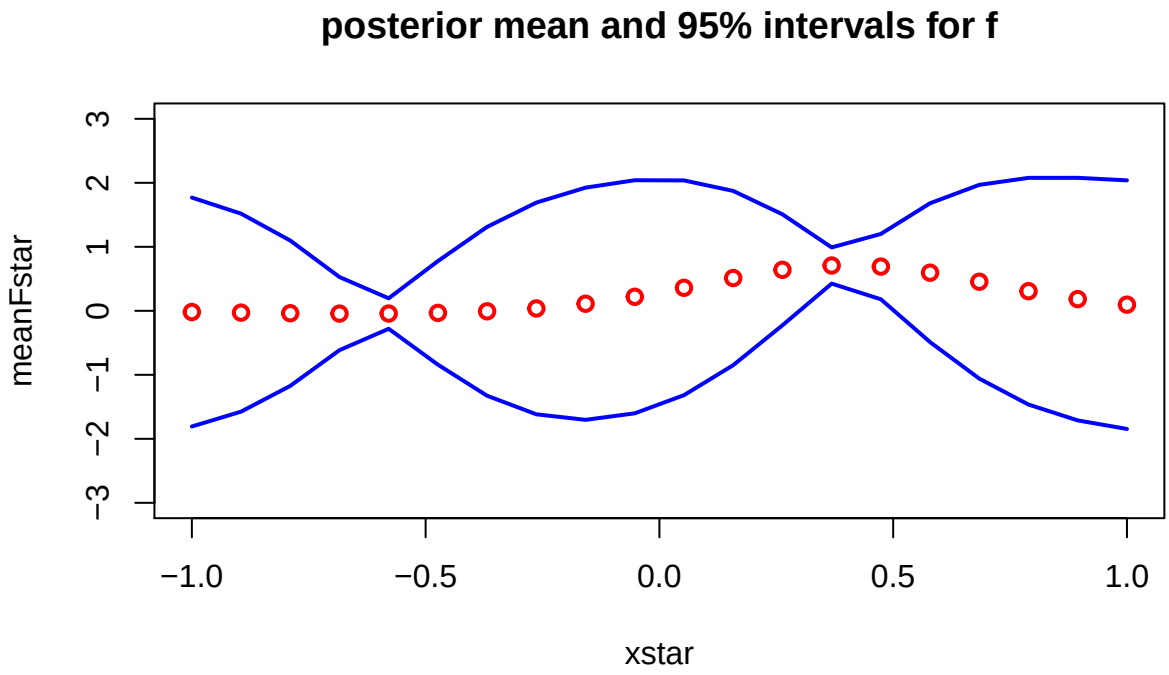
1.1 Posterior distribution of f using the squared exponential kernel

Code in appendix

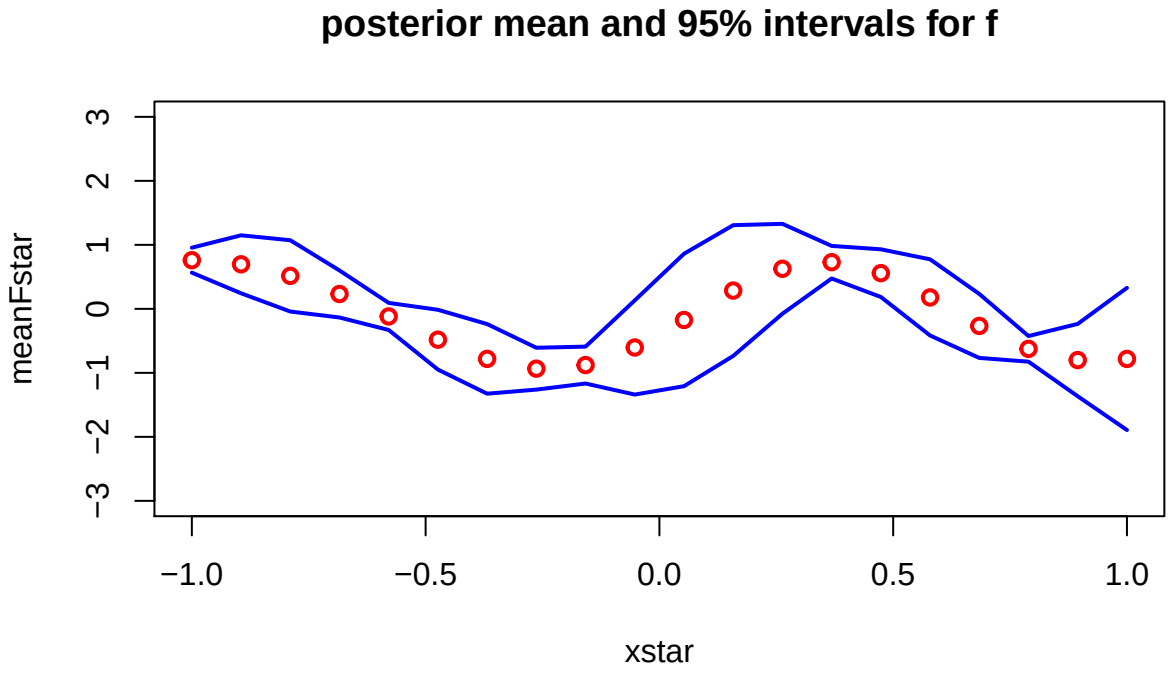
1.2 Compute posterior of f with updated prior $(x, y) = (0.4, 0.719)$ and hyperparameters $\sigma_f = 1$ and $l = 0.3$ (Assume $\sigma_n = 0.1$)



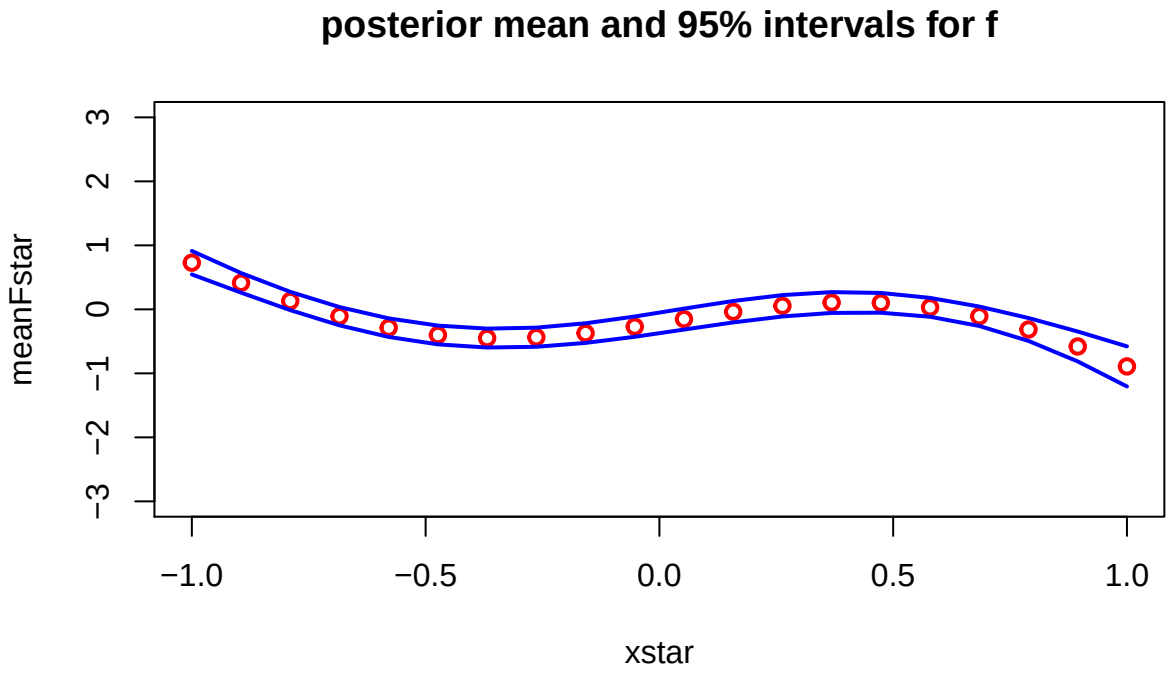
1.3 Update posterior of f with $(x, y) = (-0.6, -0.044)$



1.4 Compute posterior distribution of f using five data points



1.5 Compute posterior distribution of f using five data points by setting hyperparameter $l = 1$



We can see that increasing l has the effect of making the posterior of f more smooth. The distance between two points in the input space has to be large for us to see a significant difference in their $f*$ values.

2. GP Regression with kernlab

2.1 Square expontial kernel function

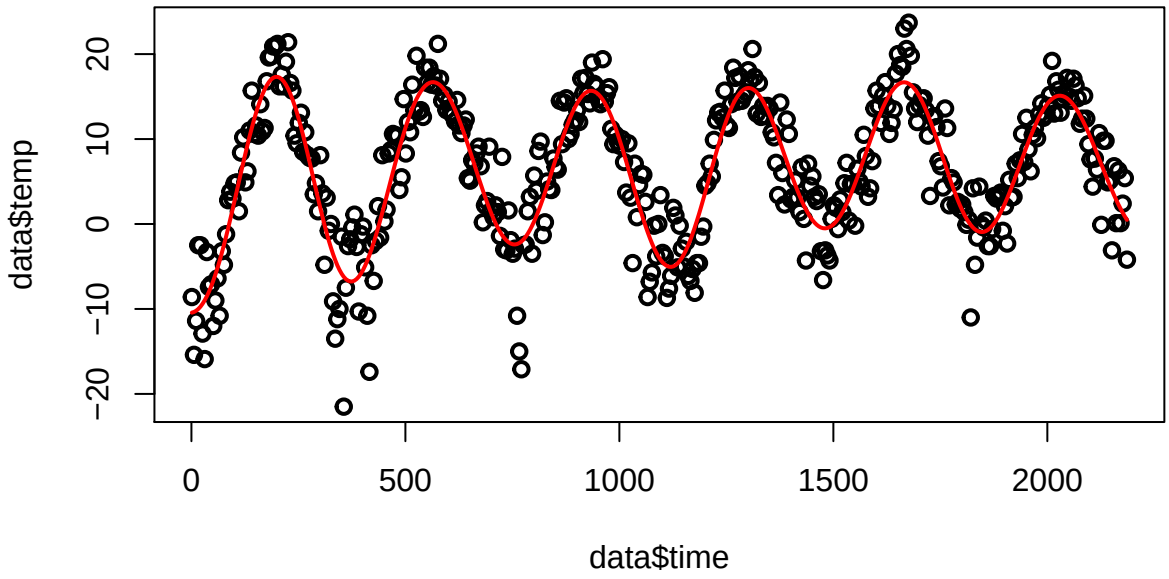
Table 1: Evaluated at (x,x')=(1,2)

0.1353353

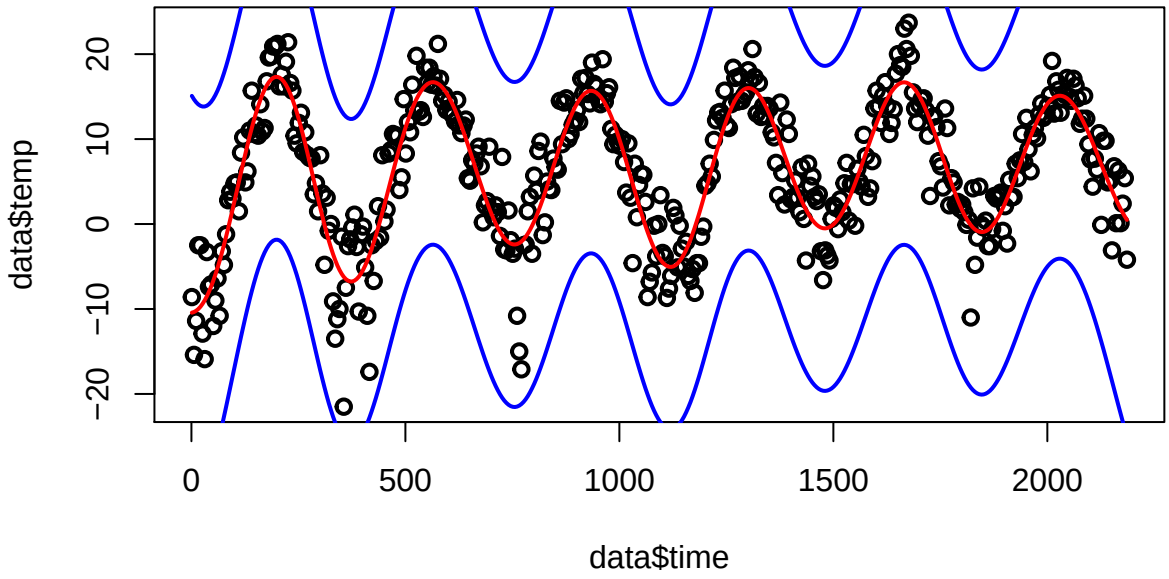
Table 2: Covariance matrix K(X, X*)

0.1353353	0.0003355	0.0000000
0.1353353	1.0000000	0.1353353
0.0003355	0.1353353	1.0000000

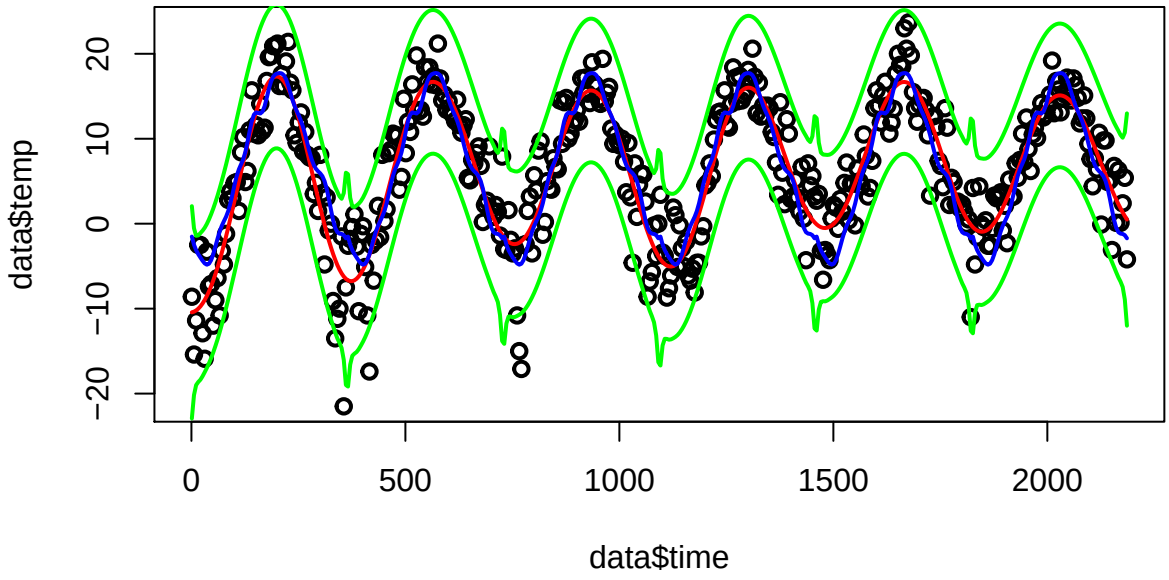
2.2 GP regression model using $time$ variable and squared exponential function with hyperparameters $sigma_f = 20$ and $l = 0.2$



2.3 Superimpose the 95% probability bands for f

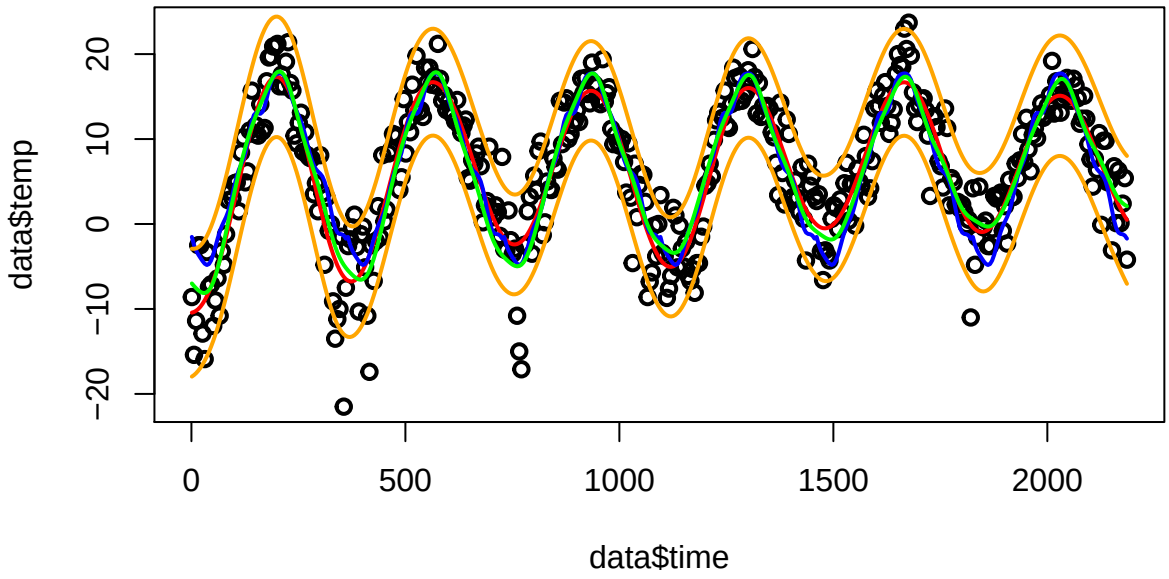


2.4 GP regression model using *day* variable and squared exponential function



We note that the time regressor gives a smoother curve for the mean predicted temperature (red curve) which seems to follows the general trend of the data quite well. In contrast, the day regressor gives a not-so-smooth curve (blue) that captures some additional variance in the temperature data that may be seasonal i.e. patterns repeating in every cycle. We also observe that the 95% prediction bands are tighter for the blue curve than the red.

2.5 GP regression model using *time* variable and generalized periodic kernel with hyperparameters $sigma_f = 20, l1 = 1, l2 = 10$ and $d = 365/sd(time)$



The mean temperature predictions using generalized periodic kernel is somewhere between the red (time regressor) and the blue (day regressor) curve. It follows the overall trend in the data while at the same time capturing some of the seasonal variance.

3. GP Classification with kernlab

3.1 GP classification using 2 covariates - `varWave` and `skewWave`

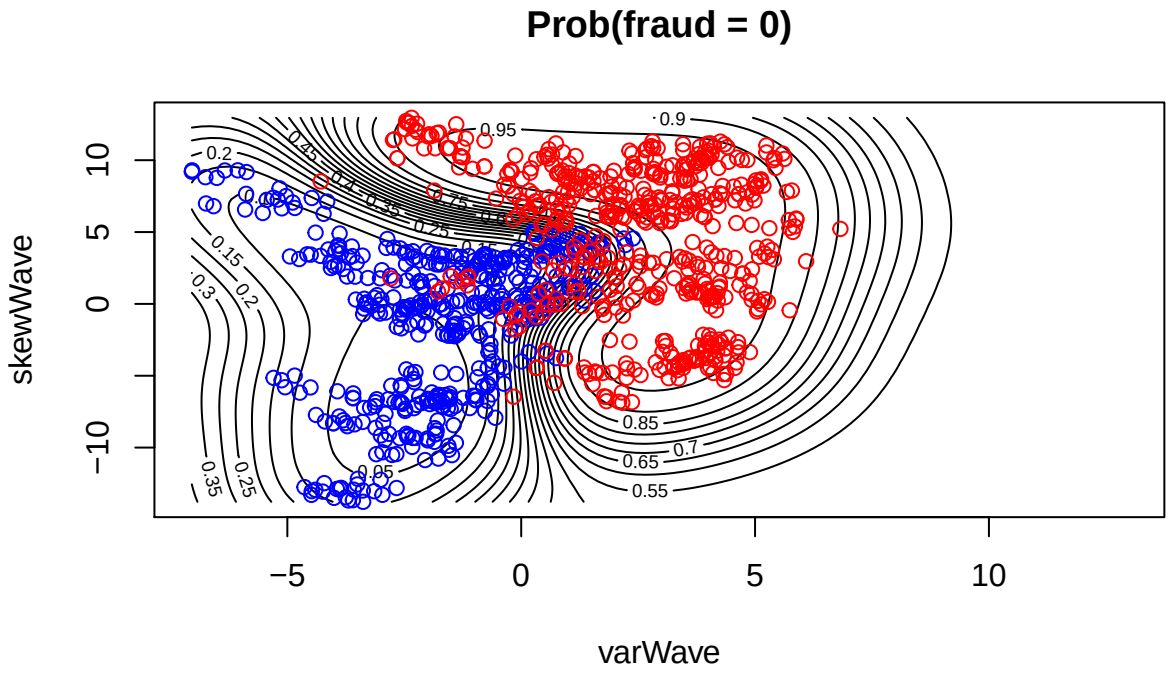
```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

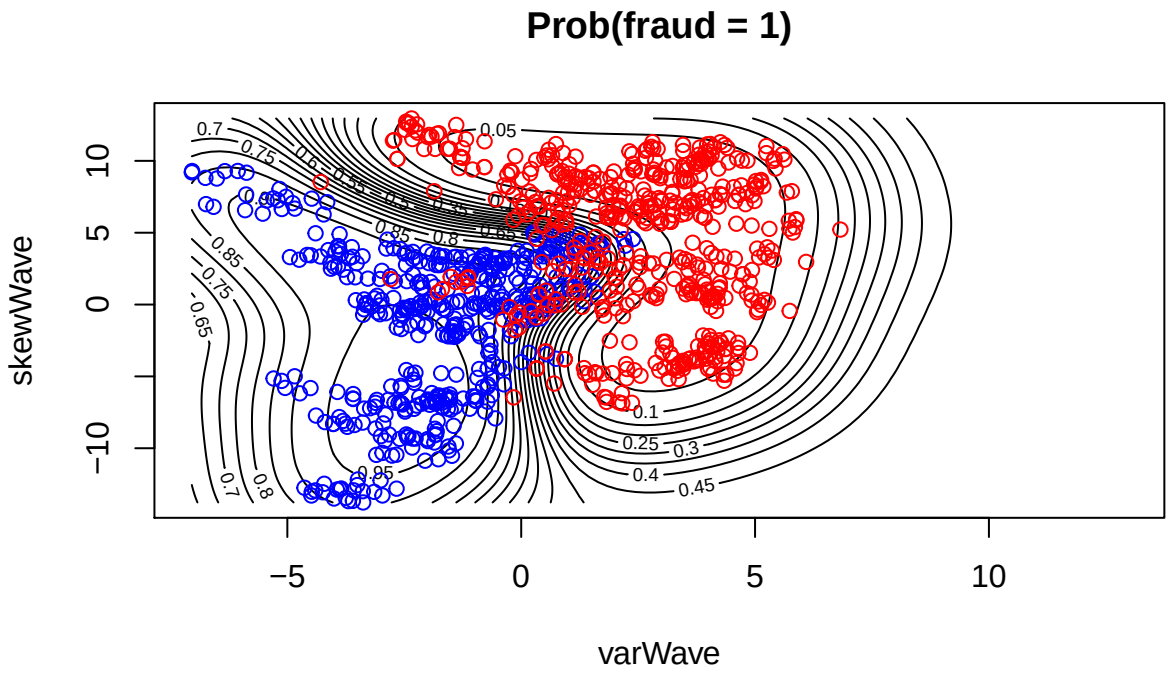
Table 3: Confusion matrix (train)

	0	1
0	503	18
1	41	438

Table 4: Classifier Accuracy (train)

x
0.941





3.2 Predictions on test set and classification accuracy of model with 2 covariates

Table 5: Confusion matrix (test)

	0	1
0	199	9
1	19	145

Table 6: Classifier Accuracy (test)

x
0.9247312

3.3 Accuracy of GP classification using 4 covariates - `varWave`, `skewWave`, `kurtWave`, `entropyWave`

```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

Table 7: Confusion matrix (test)

	0	1
0	216	0
1	2	154

Table 8: Classifier Accuracy comparison (test)

	accuracy
model with 2 covariates	0.9247312
model with 4 covariates	0.9946237

The classifier using 4 covariates has much better accuracy than the classifier using only 2. Therefore, all the 4 extracted features from images are important in detecting a fraud banknote

Appendix (Code)

```
knitr::opts_chunk$set(echo=FALSE, message=FALSE, warning=FALSE, error=FALSE)

library(kernlab)
library(AtmRay)
############################################################
# 1. Implementing GP Regression
############################################################

# Covariance function
SquaredExpKernel <- function(x1, x2, sigmaF=1, l=1){
  n1 <- length(x1)
  n2 <- length(x2)
  K  <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
```

```
  }
  return(K)
}


# Posterior GP of f
posteriorGP <- function(x, y, xstar, sigmaNoise, k, ...) {
  n <- length(x)
  kss <- k(x1=xstar, x2=xstar, ...)
  kxx <- k(x1=x, x2=x, ...)
  kxs <- k(x1=x, x2=xstar, ...)

  # Cholesky decomposition of prior marginal variance of noisy training points
  L   <- t(chol(kxx + sigmaNoise^2*diag(n)))

  # Mean vector of f*
  alpha <- solve(t(L), solve(L, y))
  meanVec <- t(kxs) %*% alpha

  # Covariance matrix of f*
  v <- solve(L, kxs)
  covMat <- kss - (t(v) %*% v)

  return(list(meanVec=meanVec, covMat=covMat))
}

# Plot mean and 95% Probability bands for f*
plotGPPosterior <- function(xstar, meanFstar, covFstar, plottitle) {
  plot(xstar, meanFstar, ylim=c(-3, 3), col="red", lwd=2)
  lines(xstar, meanFstar + 1.96*sqrt(diag(covFstar)), col="blue", lwd=2)
  lines(xstar, meanFstar - 1.96*sqrt(diag(covFstar)), col="blue", lwd=2)
  title(main=plottitle)
}

xgrid <- seq(-1, 1, length=20)
GP <- posteriorGP(x=0.4, y=0.719,
                  xstar=xgrid, sigmaNoise=0.1, k=SquaredExpKernel,
                  sigmaF=1, l=0.3)
plotGPPosterior(xgrid, GP$meanVec, GP$covMat, plottitle="posterior mean and 95% intervals for f")

# Update posterior with (x,y)=(-0.6,-0.044)
GP <- posteriorGP(x=c(0.4, -0.6), y=c(0.719, -0.044),
                  xstar=xgrid, sigmaNoise=0.1, k=SquaredExpKernel,
                  sigmaF=1, l=0.3)
plotGPPosterior(xgrid, GP$meanVec, GP$covMat, plottitle="posterior mean and 95% intervals for f")

# Compute posterior for five data points with l=0.3
GP <- posteriorGP(x=c(-1.0, -0.6, -0.2, 0.4, 0.8),
                  y=c(0.768, -0.044, -0.940, 0.719, -0.664),
                  xstar=xgrid, sigmaNoise=0.1, k=SquaredExpKernel,
                  sigmaF=1, l=0.3)
plotGPPosterior(xgrid, GP$meanVec, GP$covMat, plottitle="posterior mean and 95% intervals for f")
```

```r
# Compute posterior using five data points with l=1
GP <- posteriorGP(x=c(-1.0, -0.6, -0.2, 0.4, 0.8),
                  y=c(0.768, -0.044, -0.940, 0.719, -0.664),
                  xstar=xgrid, sigmaNoise=0.1, k=SquaredExpKernel,
                  sigmaF=1, l=1)
plotGPPosterior(xgrid, GP$meanVec, GP$covMat, plottitle="posterior mean and 95% intervals for f")

############################################################
# 2. GP Regression with kernlab
############################################################

# Read data and add variables time and day
SthlmTemp <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/
Code/TempTullinge.csv", header=TRUE, sep=";")
SthlmTemp$time <- 1:nrow(SthlmTemp)
SthlmTemp$day  <- rep(c(1:365), 6)

# Subsample the data- use only every fifth observation
data <- SthlmTemp[seq(1, nrow(SthlmTemp), 5), ]


# Square expontial kernel function
SEkernel <- function(sigmaf=1, ell=1)
{
  rval <- function(x1, x2) {
    n1 <- length(x1)
    n2 <- length(x2)
    K  <- matrix(NA, n1, n2)
    for (i in 1:n2){
      K[, i] <- sigmaf^2*exp(-0.5*( (x1-x2[i])/ell)^2 )
    }
    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}
k = SEkernel(sigmaf=1, ell=0.5)


# Evaluate in (x,x')=(1,2)
knitr::kable(k(x1=1, x2=2), caption="Evaluated at (x,x')=(1,2)")
# Covariance matrix K(X, X*)
kxs <- kernelMatrix(kernel=k, x=c(1,3,4), y=c(2,3,4))
knitr::kable(kxs, caption="Covariance matrix K(X, X*)")

# Estimate traning noise using simple quadratic regression on time variable
quadFit <- lm(temp ~ time + I(time^2), data=data)
sigmaNoise <- sd(quadFit$residuals)


# GP regression model
GPfit <- gausspr(x=data$time, y=data$temp,
                 kernel=SEkernel, kpar=list(sigmaf=20, ell=0.2),
```

```
                  var=sigmaNoise^2)
# Predicting the training data
postMean <- predict(GPfit, data$time)


# Plot data and mean predictions
plot(data$time, data$temp, lwd=2)
lines(data$time, postMean, col="red", lwd=2)

# Compute 95% prediction bands
postGP <- posteriorGP(x=scale(data$time), y=data$temp,
                      xstar=scale(data$time), sigmaNoise=sigmaNoise^2, k=SquaredExpKernel,
                      sigmaF=20, l=0.2)

# Plot 95% prediction bands
plot(data$time, data$temp, lwd=2)
lines(data$time, postMean, col="red", lwd=2)
lines(data$time, postMean + 1.96*sqrt(diag(postGP$covMat)), col="blue", lwd=2)
lines(data$time, postMean - 1.96*sqrt(diag(postGP$covMat)), col="blue", lwd=2)

# Estimate traning noise using simple quadratic regression on day variable
quadFit <- lm(temp ~ day + I(day^2), data=data)
sigmaNoise <- sd(quadFit$residuals)


# GP regression model
k = SEkernel(sigmaf=20, ell=0.2)
GPfit <- gausspr(x=data$day, y=data$temp, kernel=k, var=sigmaNoise^2)
# Predicting the training data
postMean2 <- predict(GPfit, data$day)


# Plot data and mean predictions from both GPs
plot(data$time, data$temp, lwd=2)
lines(data$time, postMean, col="red", lwd=2)
lines(data$time, postMean2, col="blue", lwd=2)
# Plot 95% prediction bands
postGP <- posteriorGP(x=scale(data$day), y=data$temp,
                      xstar=scale(data$day), sigmaNoise=sigmaNoise^2, k=SquaredExpKernel,
                      sigmaF=20, l=0.2)
lines(data$time, postMean + 1.96*sqrt(diag(postGP$covMat)), col="green", lwd=2)
lines(data$time, postMean - 1.96*sqrt(diag(postGP$covMat)), col="green", lwd=2)

# Generalized periodic kernel
PeriodicKern <- function(sigmaf=1, l1=1, l2=1, d)
{
  rval <- function(x1, x2) {
    n1 <- length(x1)
    n2 <- length(x2)
    K  <- matrix(NA, n1, n2)
    for (i in 1:n2){
      K[, i] <- sigmaf^2 * exp(-2*(sin(pi*abs(x1-x2[i])/d)/l1)^2) * exp(-0.5*((x1-x2[i])/l2)^2)
    }
```

```
    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}

# GP regression model
GPfit <- gausspr(x=data$time, y=data$temp,
                 kernel=PeriodicKern, kpar=list(sigmaf=20, l1=1, l2=10, d=365/sd(data$time)),
                 var=sigmaNoise^2)
# Predicting the training data
postMean3 <- predict(GPfit, data$time)

# Plot data and mean predictions from all GPs
plot(data$time, data$temp, lwd=2)
lines(data$time, postMean, col="red", lwd=2)
lines(data$time, postMean2, col="blue", lwd=2)
lines(data$time, postMean3, col="green", lwd=2)

# 95% prediction interval implementation
k = PeriodicKern(sigmaf=20, l1=1, l2=10, d=365/sd(data$time))
postGP <- posteriorGP(x=scale(data$time), y=data$temp,
                      xstar=scale(data$time), sigmaNoise=sigmaNoise^2, k=k)
# Plot 95% prediction bands
lines(data$time, postMean + 1.96*sqrt(diag(postGP$covMat)), col="orange", lwd=2)
lines(data$time, postMean - 1.96*sqrt(diag(postGP$covMat)), col="orange", lwd=2)

################################################################
# 3. GP Classification with kernlab
################################################################

# Read the data
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/
GaussianProcess/Code/banknoteFraud.csv", header=FALSE, sep=",")
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])

# Train-test split
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size=1000, replace=FALSE)
train <- data[SelectTraining, ]
test  <- data[-SelectTraining, ]


# GP classification model using 2 covariates
GPfit     <- gausspr(fraud ~ varWave + skewWave, data=train)
fraudPred <- predict(GPfit, train[, 1:2])

# Confusion matrix and accuracy of the classifier
confMat <- table(fraudPred, train$fraud)
acc     <- sum(diag(confMat)) / sum(confMat)

knitr::kable(confMat, caption="Confusion matrix (train)")
```

```r
knitr::kable(acc, caption="Classifier Accuracy (train)")


# Plot contours of the prediction probabilities
x1 <- seq(min(train$varWave), max(train$skewWave), length=100)
x2 <- seq(min(train$skewWave), max(train$skewWave), length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind.data.frame(c(gridPoints$x), c(gridPoints$y))
names(gridPoints) <- names(train)[1:2]

fraudProb <- predict(GPfit, gridPoints, type="probabilities")


# Plotting for Prob(fraud = 0)
contour(x1, x2, matrix(fraudProb[, 1], 100, byrow=TRUE), 20,
        xlab="varWave", ylab="skewWave", main='Prob(fraud = 0)')
points(train[train$fraud==1, 1], train[train$fraud==1, 2], col="blue")
points(train[train$fraud==0, 1], train[train$fraud==0, 2], col="red")

# Plotting for Prob(fraud = 1)
contour(x1, x2, matrix(fraudProb[, 2], 100, byrow=TRUE), 20,
        xlab="varWave", ylab="skewWave", main='Prob(fraud = 1)')
points(train[train$fraud==1, 1], train[train$fraud==1, 2], col="blue")
points(train[train$fraud==0, 1], train[train$fraud==0, 2], col="red")

# Predict on the test set
fraudPred <- predict(GPfit, test[, 1:2])

# Confusion matrix and accuracy of the classifier
confMat <- table(fraudPred, test$fraud)
acc1    <- sum(diag(confMat)) / sum(confMat)

knitr::kable(confMat, caption="Confusion matrix (test)")
knitr::kable(acc1, caption="Classifier Accuracy (test)")
# GP classification model using all 4 covariates
GPfit     <- gausspr(fraud ~ ., data=train)
fraudPred <- predict(GPfit, test[, -5])

# Confusion matrix and accuracy of the classifier
confMat <- table(fraudPred, test$fraud)
acc2    <- sum(diag(confMat)) / sum(confMat)

knitr::kable(confMat, caption="Confusion matrix (test)")
knitr::kable(data.frame(accuracy=c(acc1, acc2),
                        row.names=c("model with 2 covariates", "model with 4 covariates")),
             caption="Classifier Accuracy comparison (test)")
```